Power System Protection and Control
ISSN:1674-3415

CONCIERGE AGENT: A LOCAL HYBRID VISION-LANGUAGE FRAMEWORK FOR PRIVACY PRESERVING AUTONOMOUS COMPUTER USE

Gagan P[1], Josy Elsa Varghese[2], Bhargav Koduru[3], Anirudh C Balaji[3], Aby Binoy[4]

1,2,3,4,5 Dept. of CSE (AI & ML), Ramaiah Institute of Technology, Bengaluru, India.
Email: [1] josyvarghese@msrit.edu, [2] 1ms22ci026@msrit.edu, [3] 1ms22ci017@msrit.edu, [4] 1ms22ci009@msrit.edu, [5]1ms22ci003@msrit.edu
Orchid Id number: 20000-0002-4787-144X
Corresponding Author*: Josy Elsa Varghese

Received: 15/04/2026 **Revised**: 10/05/2026 **Accepted**: 05/06/2026

***ABSTRACT*:**

The majority of computer usage agents available today are based on cloud-computing AI models that need constant communication of screenshots and user activity with servers, which poses significant privacy risks when the task involves personal information, credentials, or documents. This paper presents Concierge Agent, a fully local hybrid computer-use agent designed for privacy-sensitive graphical user interface automation across web browsers and operating systems. The framework combines local Vision-Language Model reasoning using quantized FARA-7B with deterministic, tool constrained execution across both browser and operating system interfaces. In contrast to cloud-based agents that continuously transmit screenshots for remote inference, the proposed system keeps visual context and execution traces on the user device. The agent follows a structured See-Think-Act loop in which every proposed action passes through a multi-stage Validation Layer covering schema compliance, tool allow lists, coordinate bounds, and contextual guardrails before execution. Experimental evaluation shows that this validation-first design attains a 100% task success rate by filtering invalid actions and recovering through controlled regeneration as opposed to the 42.1% success rate for the cloud-based GPT-4o. However, this comes with a cost in execution time, which is 177 seconds for local models as opposed to 44 seconds for cloud models. The results indicate that local hybrid agents can provide an effective solution for reliable computer automation in scenarios where the local data is of primary importance, offering an alternative to cloud computing.

***Keywords**: Computer Use Agents, Vision Language Models, Local Inference, Privacy Preserving Automation, Browser Automation, Operating System Control, Gui Agents*

INTRODUCTION

Artificial intelligence has thus emerged as a revolutionary paradigm for the automation of human-computer interactions, taking computation beyond the realm of mere response generation to active environment manipulation [1]. Traditional automation technologies, such as Robotic Process Automation and script-based technologies including Selenium, have long been the basis of repetitive process automation. However, rule-based technologies are plagued with several disadvantages, as they heavily rely on selectors, coordinates, and scripts, which become obsolete when the interface changes or dynamic elements are introduced [2]. There are tremendous advancements in the way computers can automate processes, and Vision Language Models play a crucial role in this scenario. For instance, automation technologies were once based on writing strict, rigid scripts, which would fail if a single button changed positions. Today, we are living with AI technologies that look at the screen, understand what is happening, and simply act upon the scenario [3].

Recent studies have also proven the magnitude of this change by demonstrating how these new AI-based agents can surf complex websites and complete workflows with the level of flexibility that the older rule-based systems could not achieve [4]. Despite the significant progress made in the development of computer usage agents, the security and privacy of the system is a major concern when considering the deployment of such systems. Most cloud-based VLM systems involve the constant transmission of screenshots and user interfaces to inference servers [5]. This architectural choice also poses significant privacy risks when the workflows involve the use of

DOI: 10.46121/pspc.54.2.50

personal content, login credentials, financial information, or confidential documents. The second limitation of current computer usage agents is the scope of tasks

that can be executed by the system. Most state-of-the-art computer usage agents are browser-based, but realistic computing workflows also involve operations at the operating system level, including the download of files, management of directories, system configuration, and the invocation of local applications [6].

The motivation for doing this was because computer work in the present era is monotonous, and ideally, all of this should be done by an AI. Every day, people lose hours of their precious time simply searching for information, organizing their files, and filling out forms. Such tasks are perfect for an AI agent. However, currently, using an AI essentially means allowing a tech company access to your personal information. That was not our goal. We aimed to create an intelligent assistant that could adapt to user needs and perform digital work, without personal information being compromised.

The proposed research addresses the following challenges:

- Privacy exposure in cloud-based inference: The existing high capability agents rely only on remote model execution, requiring continuous screenshot transmissions that expose the sensitive user data to the external servers.
- Limited unified control across execution domains: Many web agents demonstrate strong browser navigation but they remain confined to web-only tasks, hence, they lack the capability to handle operating system level operations required for complete workflow automation.
- Reliability challenges in action execution: Coordinate-based actions and unconstrained model outputs lead to failures when the screen resolution changes, interface layouts shift, or the models generate references to unavailable elements. The contributions of this research are as follows:

1) A completely local VLM inference architecture for computer-use reasoning, leveraging quantized FARA-7B [7], without the need for external transmission of screenshots and user interface context.
2) A hybrid execution framework that leverages deterministic browser automation with limited operating-system commands, allowing end-to-end execution beyond browser-based tasks.
3) A multi-stage Validation Layer with explicit action schemas and safety mechanisms, allowing reliable task completion through conservative, checked execution.
4) Comprehensive evaluation of the tradeoff between execution latency and privacy-preserving reliability.

LITERATURE SURVEY

The field of autonomous computer use agents has been evolving rapidly from text-based conversational systems to multimodal frameworks that can interpret and interact directly with graphical user interfaces.

A. Vision-Language Model Agents

Cloud-hosted Vision Language Models have demonstrated strong capabilities in screen understanding and action planning. GPT-4o established foundational multimodal reasoning for user interface interpretation [1], while LLaVA introduced instruction tuning approaches for a vision-language interaction [3]. CogAgent presents a VLM specifically designed for GUI agents with a strong grounding capability [8]. However, the cloud deployment introduces privacy limitations as the remote inference necessitates screenshot transmissions [5].

B. Web and OS Navigation Agents

WebVoyager has introduced end-to-end web navigation using multimodal models [4], while Agent-E investigated the hierarchical task decomposition for agentic web systems [9]. Research has been accelerated by benchmarks including WebArena [10] and Mind2Web [11]. For OS-level automation, the AGENT S investigates human-like computer operation [12], OSWorld provides a cross-platform evaluation [6], Windows Agent Arena offers a Windows-specific benchmark [13], and the ScreenAgent demonstrates VLM-driven control with planning-acting-reflecting phases [14].

C. Tool-Augmented Paradigms

The SWE agent emphasized a structured, tool-based interface over unrestricted shell access, demonstrating that constraining agents to a well-defined tool interfaces reduce the failure modes [15]. SeeClick addresses GUI grounding challenges through element localization benchmarks [16], and these approaches inform the Validation Layer design in our proposed framework.

DOI: 10.46121/pspc.54.2.50

From the literature, we see three gaps emerging: (i) lack of privacy-preserving local execution in capable multimodal agents, (ii) limited unified control across browser and OS domains, and (iii) reliability challenges under dynamic interface conditions. Table 1 presents a comparative analysis positioning our proposed Concierge Agent framework.

Therefore, to address these challenges, the proposed framework integrates a local VLM inference with a hybrid, tool-constrained execution:

- Local inference using quantized FARA-7B eliminates the requirement for an external data transmission, ensuring that screenshots and decision-making processes remain entirely on the device.
- Our hybrid execution layer combines browser automation through Playwright with a restricted set of operating system commands, enabling workflows that span web interaction and local system operations.
- Our multi-stage Validation Layer reduces failure modes by requiring a model's outputs to conform to the explicit schemas, and validation enforcing allowlists and contextual bounds before execution.

PROPOSED METHODOLOGY

This section presents our methodology for the design and implementation of the Concierge Agent. Our methodology enables an autonomous GUI interaction while preserving user privacy through a structured control mechanism as detailed in the following steps.

A. See-Think-Act Control Loop

Our proposed system operates through an iterative See-Think-Act control loop that enables a continuous interaction between perception, reasoning, and execution components. Each iteration proceeds through three distinct phases:

See Phase: The perception module captures a high-resolution screenshot (1920×1080) and constructs a contextual prompt. This prompt combines the visual observation encoded as a base64 image, the original user task description, and a compact trace of the k most recent actions with their outcomes. The trace provides a temporal context enabling the VLM to understand the execution progress and avoid any repeated failures during this process.

Think Phase: The locally hosted FARA-7B analyzes the visual context. The model then receives a system prompt specifying the action schema and available tools, followed by a constructed context, and generates a structured action description. Inference occurs entirely on device using a 4-bit quantized weights, ensuring zero data transmission; then the model outputs a JSON-formatted action conforming to the predefined schema.

Act Phase: In this phase, the generated action undergoes a multi-stage validation before execution. Valid actions are then dispatched to the appropriate execution tool, either by browser automation or by an OS command interface. The resulting screen state is then captured for the next iteration. Invalid actions trigger rejection with a descriptive reason, prompting the VLM to regenerate with corrective feedback.

B. Action Schema Design

To reduce ambiguity and parsing errors, the system restricts the action generation to a compact schema; each action contains an action_type field that is selected from six primitives:

- click(x, y): Mouse click at screen coordinates (x, y) within bounds [0, W] × [0, H] where W = 1920 and H = 1080.
- type(text): Keyboard input of the specified text string to the currently focused element.
- scroll(direction, amount): Vertical or a horizontal scroll with a specified magnitude.
- navigate(url): Browser navigation to the specified URL.
- os_cmd(command): Execution of a restricted operating system command from the allowlist.
- done(status): Task completion signal with success or failure indication. This constrained schema thus ensures our model outputs map to well-defined execution primitives, eliminating ambiguous natural language instructions.

C. Multi Stage Validation Layer

The Validation Layer is the architectural component that makes the reliable execution with the local model feasible. Each proposed action goes through four validation layers before execution:

DOI: 10.46121/pspc.54.2.50

Stage 1 - Schema Compliance (V1): This checks whether the action meets the JSON schema requirements, ensuring all necessary parameters are included and of the correct data type. Malformed output or parameters immediately trigger rejection.
Stage 2 - Tool Allowlist (V2): This checks the "os_cmd' action type and verifies whether the command is part of the safe set {ls, cd, pwd, cat, mkdir, cp, mv, open}. If the command is not part of this set, the action is rejected.
Stage 3 - Bounds Checking (V3): This checks the "click' action type and verifies whether the x and y coordinates are within the range 0 <= x <= 1920 and 0 <= y <= 1080. If the action is out of bounds, the action is rejected.
Stage 4 - Contextual Guardrails (V4): This checks the execution patterns for failure indicators such as loop detection (consecutive repetitions of the same action three times); high retries (five or more failed attempts); and contradiction detection (contradictory actions in sequence).

The combined validation function is: $V(a) = V1(a) \wedge V2(a) \wedge V3(a) \wedge V4(a, H)$

When validation is unsuccessful, the system adds the reason for the failure to the context and prompts the VLM to regenerate the task. This safe approach may cause extra iterations, but it ensures the task is always performed without execution errors.

D. Algorithm Specification

Algorithm 1 specifies the See-Think-Act cycle with integrated validation. The cycle ends when the task is done, the maximum number of iterations is reached, or an unrecoverable failure occurs.
Algorithm 1. See-Think-Act Control Loop Require: Task description T, maximum iterations N

Ensure: Execution history H
1: Initialize history H ← empty, iteration i ← 0 2: while i < N do
3: I← CaptureScreen()
4: C ← BuildContext(T, I, H) 5: a ← VLM.ProposeAction(C) 6: if Validate(a) then
7: r ← Execute(a)
8: Append(a, r) to H
9: if CheckDone(T, I, H) then break
10: end if
11: else
12: Append (a, rejected) to H 13: a ← VLM.Retry(C, reason) 14: end if
15: i ← i + 1
16: end while
17: return H

Table 1. Comparative Analysis of Representative Computer Use Agents and Related Systems

System	Primary Focus	Local	OS	Action Style	Key Limitations
GPT-4V Agents [1]	Multimodal reasoning	No	Limited	Generated actions	Privacy exposure; network dependency
WebVoyager [4]	Web navigation	Cloud	No	Web actions	Browser-only; no OS workflows
Agent-E [5]	Web workflows	Cloud	No	Hierarchical actions	Browser-bound execution
LLaVA [3]	General VLM	Yes	No	Conversational	Not action-native
CogAgent [14]	GUI grounding	Varies	No	Grounded UI actions	High computational requirements
ScreenAgent [13]	UI grounding	Varies	No	Grounded actions	Pipeline complexity
AGENT S [6]	Human-like use	Cloud	Yes	Perception-driven	Orchestration complexity
SWE-agent [8]	Tool-use agents	Varies	Partial	Tool calls	Integration complexity

DOI: 10.46121/pspc.54.2.50

Power System Protection and Control
ISSN:1674-3415

OSWorld [11]	Benchmark	Yes	Yes	Evaluation suite	Not an execution agent
WebArena [9]	Benchmark	Yes	No	Task suites	Benchmark focus only
Concierge Agent	**Browser + OS**	**Yes**	**Yes**	**Structured + tools**	**Local latency**

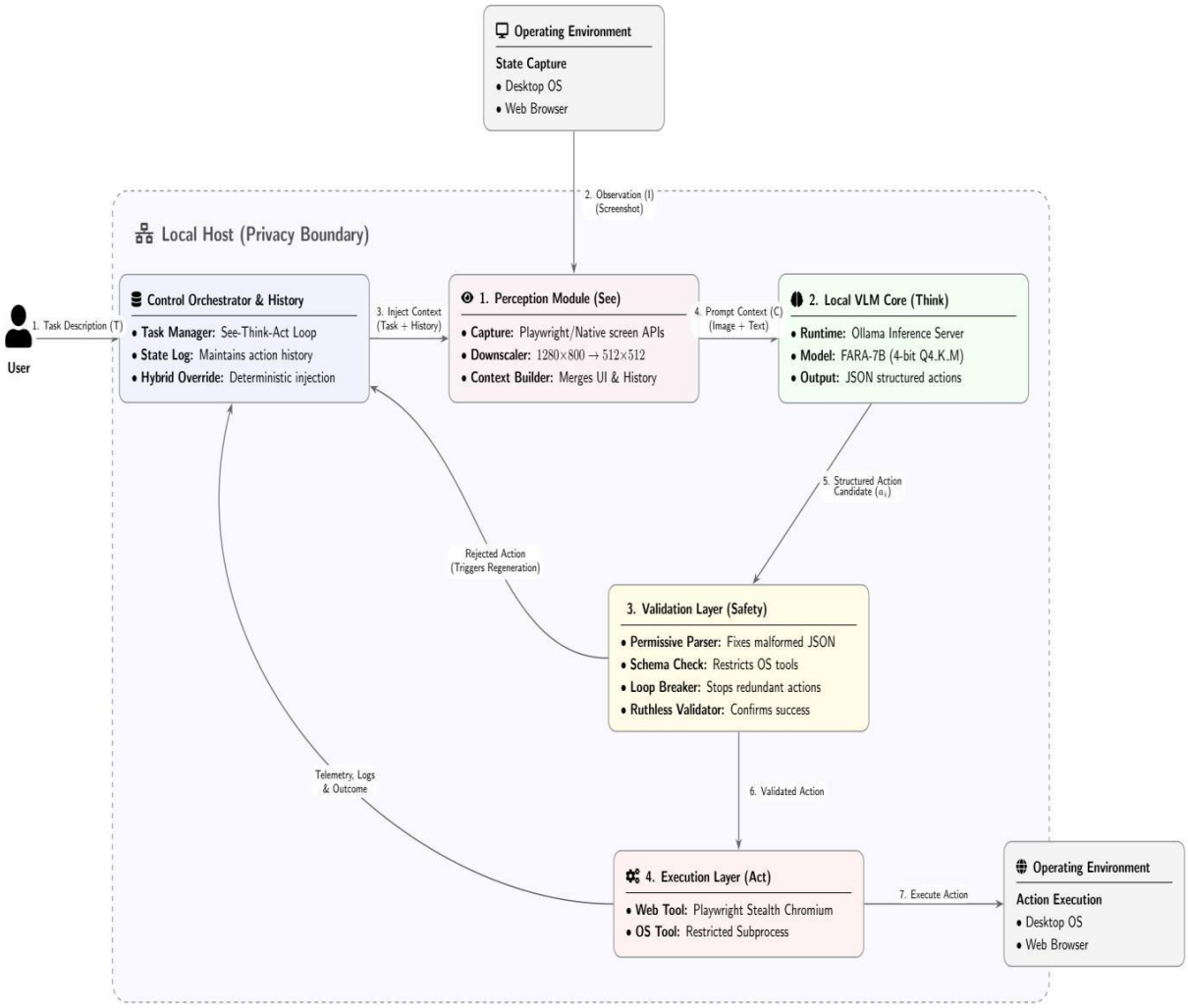

Figure 1. System architecture of Concierge Agent illustrating the See Think Act loop with local FARA-7B inference, multi stage Validation Layer, and hybrid execution. All processing occurs within the local privacy boundary.

SYSTEM ARCHITECTURE DESIGN

The proposed architecture employs a modular design supporting autonomous computer use while maintaining reliability and privacy. Figure 1 illustrates the component organization within the local processing boundary.

A. Control Orchestrator

The central orchestrator coordinates all system components and manages the execution lifecycle. It maintains the task state, including the current goals, iteration count, and the execution history. The orchestrator implements a retry policy allowing up to three regeneration attempts per action before escalating to error recovery. Termination conditions include successful done action, maximum iteration limit (default N = 50), or detection of unrecoverable failure states.

DOI: 10.46121/pspc.54.2.50

B. Perception Module

The perception module interfaces with a display server to capture screenshots via the platform-native APIs. On Ubuntu 22.04, it utilizes gnome-screenshot for desktop capture and Playwright's built-in capture for the browser content. Screenshots are resized to 1920×1080 resolution and encoded as a base64 PNG for inclusion in the VLM prompt. The module maintains a record of the most recent actions and formats them as structured data, including information about the action type, parameters, and outcome.

C. Local VLM Core

Currently, the reasoning component employs the quantized version of the FARA-7B model, running on the Ollama inference runtime environment. Quantization of the model has been done to 4-bit precision, employing the GGUF format, resulting in an approximate 4 GB model size while preserving the model's ability to understand visuals. The inference occurs on the CPU with 16 GB RAM, taking around 3 to 5 seconds per forward pass.

The VLM receives the system prompt, which includes the action schema, followed by the user task and the visual context. The output parsing retrieves the JSON structure of the action, with an option to resort to regex in the event of an improper response.

D. Execution Layer

The execution layer dispatches validated actions to deterministic tools:
Browser Automation Tool: This implements web interaction through Playwright with a Chromium backend. Supports programmatic navigation, coordinate-based clicking with automatic coordinate transformation, text input with proper focus management, and scroll operations. The browser runs in headed mode at 1920×1080 resolution to match the perception module's capture dimensions.

OS Command Tool: Executes allowlisted system commands through a restricted subprocess interface. These commands are validated against the allowlist before its execution, with output captured for inclusion in the action history; the tool operates with user level permissions, preventing privileged operations

EXPERIMENTAL SETUP

A. System Configuration

The proposed system has been experimented on consumer-grade hardware, as follows: Operating System - Ubuntu 22.04 LTS; Hardware: Intel Core i5 processor with four cores running at 2.4 GHz, 16 GB RAM, Integrated Graphics. The FARA-7B runtime was utilized with Ollama v0.1.27, featuring 4-bit quantization. Playwright 1.40 with Chromium at 1920x1080 resolution.

B. Evaluation Tasks

The system was tested for multi-step, UI-driven workflows. The test cases were as follows:
Case A: Information Navigation. Here, the proposed agent would interpret the goal, detect UI elements from intermediate states, and evaluate success. This case tested planning from multiple observations.
Case B: E-commerce interaction. In this case, the proposed agent would search for an item, select visually similar items, and evaluate success. This case tested grounded action selection

C. Evaluation Metrics

Task Success Rate (SR): A binary completion metric that is independent of efficiency.
Task Score: Cumulative quality with penalties for inefficiency.
Latency: The end-to-end execution time. Completion: The proportion of sub-goals satisfied. Efficiency: Penalties for redundant actions.

DOI: 10.46121/pspc.54.2.50

RESULTS AND ANALYSIS

A. Task Execution Evaluation

Figure 2 shows a representative multi-step execution with the See-Think-Act cycle, along with recovery due to validation.

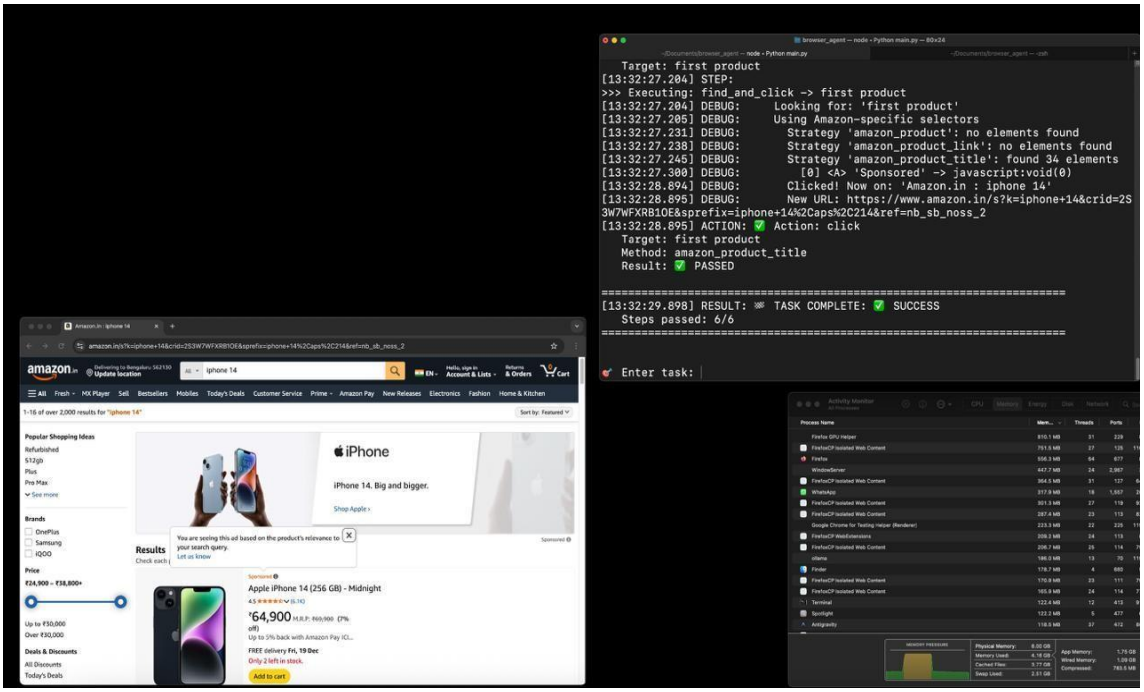

Figure 2. Execution snapshot demonstrating e-commerce navigation. The agent completes search, filtering, and selection, verifying success through execution logs and resulting UI state.

B. Benchmark Performance

Table 2 displays the performance metric between the local configuration and the cloud-hosted models. Figure 3 illustrates the performance across different evaluation dimensions.

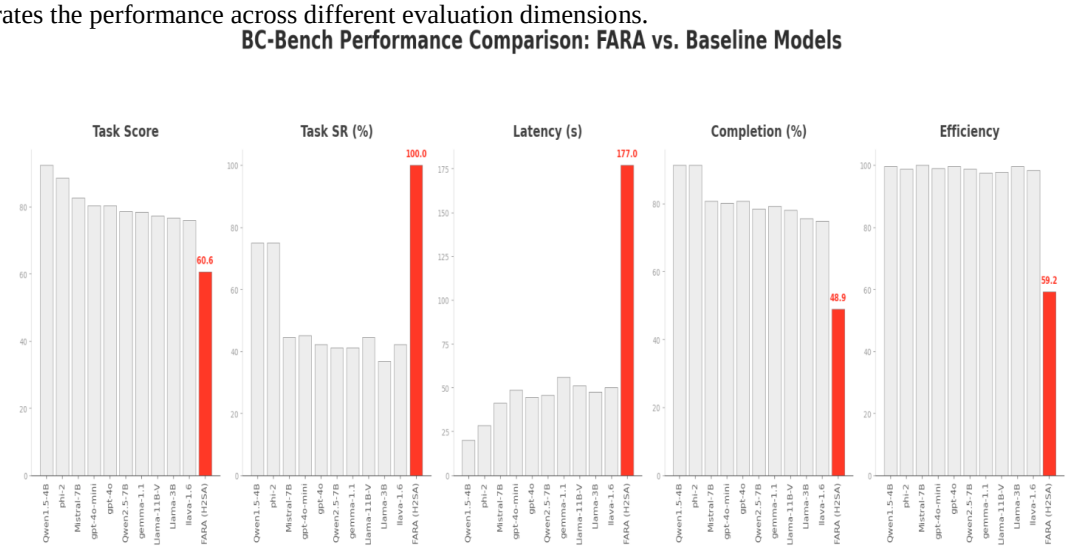

Figure 3. BC-Bench performance visualization: The proposed configuration achieves 100% task success rate despite higher latency.

DOI: 10.46121/pspc.54.2.50

Table 2. Aggregated Benchmark Performance Comparison

Rank	Model	Task Score	Task SR	Latency (s)	Completion	Eff./Speed
1	Qwen/Qwen1.5-4B-Chat	92.35	75.0%	19.95	91.3%	99.5/84.2
2	microsoft/phi-2	88.53	75.0%	28.32	91.2%	98.8/66.4
3	mistralai/Mistral-7B-Instruct	82.61	44.4%	41.43	80.7%	100.0/61.2
4	gpt-4o-mini	80.28	45.0%	48.63	80.2%	99.0/52.5
5	gpt-4o	80.20	42.1%	44.45	80.7%	99.5/50.0
6	Qwen/Qwen2.5-7B-Instruct	78.65	41.2%	45.79	78.4%	98.8/48.9
7	google/gemma-1.1-7b-it	78.31	41.2%	55.93	79.1%	97.6/47.3
8	Llama-3.2-11B-Vision-Instruct	77.15	44.4%	51.16	78.1%	97.8/44.0
9	Llama-3.2-3B-Instruct	76.74	36.8%	47.67	75.6%	99.5/45.5
10	llava-v1.6-vicuna-7b	75.95	42.1%	49.97	74.9%	98.4/44.8
11	**FARA-7B (local + validated)**	**60.56**	**100.0%**	**177.00**	**48.9%**	**59.2/41.9**

C. Analysis: Explaining the 100% Success Rate

The local configuration records a perfect Task Success Rate of 100%, as opposed to GPT-4o's 42.1%, although the local configuration records a lower Task Score of 60.56 compared to the cloud-based model of 80.20. This is due to the nature of the two metrics used to evaluate the performance of the models. Task Success Rate focuses on the binary nature of task completion irrespective of the efficiency involved in task execution, whereas the Task Score focuses on the additional steps and time involved. The local configuration records a perfect Task Success Rate due to the nature of the Validation Layer, which ensures that no fatal error occurs during task execution. When the VLM suggests invalid actions, the validator rejects the action and requests a regenerated action. The agent then retries until the action is successful.

D. Latency Analysis: The Privacy Trade-off

A 177-second latency is considerably higher than cloud-based options, which have a 44–50-second latencies. This overhead represents the penalty for maintaining a tight privacy boundary. In a workflow with sensitive information, the risk of data leakage is intolerable. The cloud agents send every screenshot outside the system, hence, our proposed system offers a 0-transmission guarantee by design. This latency is also reducible with NPUs, GPUs, or even through quantization.

LIMITATIONS AND FUTURE WORK

Local Inference Latency: The CPU-bound inference has an iteration time of several seconds. Future work includes optimized runtimes, distillation, and a scale deployment as a SaaS product.

Constrained OS Capabilities: The allow list restrictions impose limitations on privileged operations. Expansion of the safe command set is planned.

Dynamic UI Variability: The ambiguity of the grounding of the cookie banners and popups is an issue. An OCR solution is a possible improvement.

Goal Verification: The ambiguity of the end states is an issue that necessitates nuanced judgment, with confidence-based thresholds as future work.

DOI: 10.46121/pspc.54.2.50

CONCLUSION

This work presents the concept of the Concierge Agent, a completely local hybrid computer-use agent that utilizes quantized FARA-7B reasoning with deterministic, tool-constrained execution in browser and operating system interfaces. The key contribution is the multi-stage Validation layer, which ensures that task completion is possible with a 100% success rate, even with the smaller local model. This is remarkable given the lower data locality of the model itself. The 177-second average latency is the computational penalty of maintaining the privacy boundary with zero external transmission of visual data at any time during execution. The research demonstrates that successful autonomous computer use does not require the sacrifice of data locality, as local hybrid agents provide a dependable solution through the modular design of the multi-stage validation layer. The execution layer is a key contribution addressing the current gap in cloud-based solutions, enabling end-to-end workflows representative of real-world computer tasks. The results of this research position the local hybrid agents as a viable alternative to cloud-based solutions, especially as edge hardware evolves with NPUs that improve quantization, reducing the latency constraints over time while maintaining the absolute privacy benefits.

REFERENCES

1. Openai, Sally A, Gerardo A, Rubaid A, Max B, Matthew B, et al. GPT-4V(ision) System Card. Figshare. Figshare (United Kingdom); 2024.
2. Nguyen DD, Chen J, Wang Y, Wu G, Park N, Hu Z, et al. GUI Agents: A Survey. 2025 Jan 1 [cited 2026 Mar];22522. Available from: https://doi.org/10.18653/v1/2025.findings-acl.1158
3. Lee YJ, Li C, Liu H, Wu Q. Visual Instruction Tuning. 2023 Jan 1 [cited 2026 Mar];34892. Available from: https://doi.org/10.52202/075280-1516
4. He H, Yao W, Ma K, Yu W, Dai Y, Zhang H, et al. WebVoyager: Building an End-to-End Web Agent with Large Multimodal Models. 2024 Jan 1 [cited 2026 Mar];6864. Available from: https://doi.org/10.18653/v1/2024.acl-long.371
5. Sager P, Meyer B, Peng Y, Wartburg-Kottler R von, Etaiwi L, Enayati A, et al. A Comprehensive Survey of Agents for Computer Use: Foundations, Challenges, and Future Directions. ArXiv.org [Internet]. 2025 Jan 27 [cited 2026 Mar]; Available from: http://arxiv.org/abs/2501.16150
6. Cao R, Chen J, Cheng Z, Hua TJ, Lei F, Li X, et al. OSWorld: Benchmarking Multimodal Agents for Open-Ended Tasks in Real Computer Environments. 2024 Jan 1 [cited 2026 Mar];52040. Available from: https://doi.org/10.52202/079017-1650
7. Ahmed A, Yash L, Raghav M, Hussein M, Akshay N, Yash P, et al. Fara-7B: An Efficient Agentic Model for Computer Use. arXiv (Cornell University) [Internet]. 2025 Nov 24 [cited 2025 Nov]; Available from: https://doi.org/10.48550/arxiv.2511.19663
8. Hong W, Wang W, Lv Q, Xu J, Yu W, Ji J, et al. CogAgent: A Visual Language Model for GUI Agents. 2024 Jun 16 [cited 2026 Mar];14281. Available from: https://doi.org/10.1109/cvpr52733.2024.01354
9. Abuelsaad T, Akkil D, Dey P, Jagmohan A, Vempaty A, Kokku R. Agent-E: From Autonomous Web Navigation to Foundational Design Principles in Agentic Systems. arXiv (Cornell University) [Internet]. 2024 Jul 17 [cited 2026 Feb]; Available from: http://arxiv.org/abs/2407.13032
10. Zhou S. WebArena: A Realistic Web Environment for Building Autonomous Agents. In: International Conference on Learning Representations (ICLR). 2024.
11. Chen S, Deng X, Gu Y, Stevens S, Yu S, Sun H, et al. Mind2Web: Towards a Generalist Agent for the Web. 2023 Jan 1 [cited 2026 Mar];28091. Available from: https://doi.org/10.52202/075280-1220

DOI: 10.46121/pspc.54.2.50

12. Agashe S, Han JL, Gan S, Yang J, Li A, Wang XE. Agent S: An Open Agentic Framework that Uses Computers Like a Human. arXiv (Cornell University) [Internet]. 2024 Oct 10 [cited 2026 Feb]; Available from: http://arxiv.org/abs/2410.08164
13. Bonatti R, Zhao D, Bonacci F, Dupont D, Abdali S, Li Y, et al. Windows Agent Arena: Evaluating Multi-Modal OS Agents at Scale. arXiv (Cornell University) [Internet]. 2024 Sep 12 [cited 2026 Feb]; Available from: http://arxiv.org/abs/2409.08264
14. Niu R, Li J, Wang S, Fu Y, Hu X, Leng X, et al. ScreenAgent: A Vision Language Model-driven Computer Control Agent. arXiv (Cornell University) [Internet]. 2024 Feb 9 [cited 2026 Mar]; Available from: http://arxiv.org/abs/2402.07945
15. Jimenez C, Lieret K, Narasimhan K, Press O, Wettig A, Yang J, et al. SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering. 2024 Jan 1 [cited 2026 Mar];50528. Available from: https://doi.org/10.52202/079017-1601
16. Cheng K, Sun Q, Chu Y, Xu F, Li Y, Zhang J, et al. SeeClick: Harnessing GUI Grounding for Advanced Visual GUI Agents. 2024 Jan 1 [cited 2026 Mar];9313. Available from: https://doi.org/10.18653/v1/2024.acl-long.505

DOI: 10.46121/pspc.54.2.50